(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 948 854 B1

(12) **EUROPEAN PATENT SPECIFICATION**

(45) Date of publication and mention of the grant of the patent:
24.07.2002 Bulletin 2002/30

(21) Application number: **97953561.4**

(22) Date of filing: **30.12.1997**

(51) Int Cl.[7]: **H04L 12/46**, H04L 29/06, G06F 17/30, G06F 12/10

(86) International application number:
PCT/US97/24218

(87) International publication number:
WO 98/29986 (09.07.1998 Gazette 1998/27)

(54) **HASH-BASED TRANSLATION METHOD AND APPARATUS WITH MULTIPLE LEVEL COLLISION RESOLUTION**

HASHNUMMERÜBERSETZUNGS-VERFAHREN UND -GERÄT MIT MEHREREN KOLLISIONSAUFLÖSUNGSSTUFEN

PROCEDE ET DISPOSITIF DE TRADUCTION BASES SUR LE HACHAGE ET DOTES D'UNE RESOLUTION DE COLLISION A NIVEAUX MULTIPLES

(84) Designated Contracting States:
DE FR GB

(30) Priority: **30.12.1996 US 775091**

(43) Date of publication of application:
13.10.1999 Bulletin 1999/41

(73) Proprietor: **CABLETRON SYSTEMS, INC.**
Rochester, NH 03867 (US)

(72) Inventors:
- **SPINNEY, Barry, A.**
Wayland, MA 01778 (US)
- **POOLE, Nigel, T.**
Natick, MA 01760 (US)

(74) Representative: **Driver, Virginia Rozanne et al**
Page White & Farrer
54 Doughty Street
London WC1N 2LS (GB)

(56) References cited:
US-A- 5 293 595
US-A- 5 414 704

EP 0 948 854 B1

Printed by Jouve, 75001 PARIS (FR)

Description

FIELD OF THE INVENTION

[0001] This invention relates, in general, to translation methods and apparatus using table lookup and specifically to collision resolution in hash-based lookup tables.

BACKGROUND OF THE INVENTION

[0002] Translation between data formats is a common problem in data processing systems. Often the operation of the data processing system is heavily dependent on the speed and efficiency of the translation process. For example, one common example is the translation of stored binary data into a form suitable for transmission over a data network.

[0003] A data network typically consists of several nodes, at which switches are located, connected together by data transport media. A common method of transmitting data over such a network is to exchange messages between the switches, which messages comprise discrete "packets" of data. These packets can be transported over the media with one of a variety of transport techniques. In applications utilizing packetized data, data to be transported is first broken up into discrete pieces or packets which are then transmitted through the switches and reassembled at a destination. In accordance with current packet protocol, each packet generally consists of a header and an information field. The header contains the information used to transport the cell from one node to the next. The packet data is contained in the information field.

[0004] In such networks, the data source and destination are often identified by unique numerical addresses and in accordance with many popular protocols, to insure that the network can be reasonably large, the address information in the header field of such system is also often large - on the order of 5-6 bytes or 40-48 bits. A 48-bit address field can identify 2^{40} or about 300 trillion unique addresses. However, most networks have on the order of a few thousand different terminals so that only a small fraction of the potential address pool is actually needed.

[0005] The disparity in the potential number of addresses and the number of actual addresses causes inefficient use of resources. For example, if provisions were made to store each unique address, a large memory would be necessary, but only a few memory locations would be used to store the actual addresses. Therefore, for efficient use of resources at each local node, smaller address fields, conventionally called forwarding addresses, are used and a translation is made between the larger address field used in the packet headers and the smaller address field used within each local node.

[0006] A conventional translation method for translating addresses in packet switching systems uses a database in which the local, smaller addresses are stored. The larger addresses in the data packet headers are used to search the local database and retrieve the forwarding addresses. There are several well-known techniques for performing this search. One method is to use a binary tree search. Another method is to use a content addressable memory. Still other methods rely on hashing techniques to generate an index into the database. These address translation methods are described in U.S. Patent Nos. 4,587,610; 4,993,937; 5,027,350 and 5,136,580.

[0007] An important design objective in packet switching networks is controlling the flow of packets so that packets will not arrive at communication links or switches at a faster rate than they can be processed and forwarded to the next destination. If the packets cannot be processed rapidly, packet buffers may become full and packet loss will occur severely reducing the efficiency of the network. As a result, binary tree searches present difficulties in such networks. For example, a forwarding address database which holds about sixteen thousand forwarding addresses (a 16K memory) requires a 14-bit address. In the worst case, with a binary searching technique, such a memory could require 14 reads in order to locate a particular address in the memory. This number of reads is prohibitive from a performance standpoint using commonly available semiconductor memories.

[0008] Content addressable memories require only one read to compare a packet address with all stored addresses. However, content addressable memories are complex and expensive and, thus, suitable only for systems in which the forwarding database is small.

[0009] Another alternative is to use a hashing technique to convert the large number of packet header addresses into a smaller number of local forwarding addresses. Conventional hashing techniques use a mathematical transformation to map each of the packet addresses into one of a set of index addresses which are used to index into the forwarding address database. Since a large number of addresses are being reduced into a much smaller number of addresses, some of the packet addresses will inevitably map into the same index address causing a "hash collision." Some technique must be used to resolve these hash collisions so that a unique index address can be generated from each packet address.

[0010] One problem with these hashing techniques is that the efficiency of the method has been heavily dependent on the hashing transformation used in the procedure. Many hashing transformation suffer from poor memory usage, low speed and large worst-case delays, thereby making them unsuitable for use in the packet switching address trans-

lation systems. In addition, many hashing techniques utilize special circuitry to resolve collisions and are thus expensive and complicated.

[0011] Still other prior art techniques use a combination of the above-described techniques to perform the required translation. For example, U.S. Patent No. 5,414,704, the disclosure of which is hereby incorporated by reference, uses a programmable hashing technique to generate the forwarding address table indices and a binary search technique to handle hash collisions. In accordance with the disclosed method, the data packet addresses are transformed by a hash transformation into indices for a hash table. The hash table includes as each table entry, a pointer to a set of forwarding table indices arranged in a small binary tree. Hash collisions, which cause two or more data packet addresses to transform to the same hash table entry, are handled by performing a binary tree lookup to obtain a unique forwarding address table index. Since the binary tree is small (typically three levels or seven total entries) the number of reads required to traverse the tree is also small thereby significantly reducing the lookup time. A content-addressable memory (CAM) is provided to handle the cases in which more than seven data packet addresses hash to the same hash table entry.

[0012] While aforementioned combination arrangement offers a compromise in performance and cost, there are many address combinations in which the CAM must be relatively large because a significant number of data packets addresses transform to the same hash table entry. The cost of the system is therefore increased.

[0013] Accordingly, there is a need for a translation method and apparatus which efficiently uses system resources, performs well and has a lower cost than presently-available translation or lookup systems.

SUMMARY OF THE INVENTION

[0014] In accordance with the principles of the invention, a translation is performed by using a programmable hashing technique on an input number to generate a hashed number. A subset of the hashed number bits are used to index a first hash table. In first hash table locations where a hash collision does not occur, the first hash table entry contains an index into an output table which contains the desired translated output number.

[0015] In first hash table locations where a hash collision occurs, the first hash table entry contains a pointer to a first resolution table area in a second hash table. The first resolution table area contains entries which are indexed by additional bits selected from the hashed number in accordance with a mask field in the first hash table location. In the first resolution area, in each entry where a collision does not occur, the first resolution table entry contains an index into the output table which contains the desired translated output number. In each first resolution table entry where a collision does occur, the entry contains a pointer to a second resolution area in a third hash table. The second resolution table area contains entries which are indexed by still more bits selected from the hashed number in accordance with a mask field in the first resolution table location. The entries in the second resolution table may be indices into the output table or additional pointers to still further resolution tables, if hash collisions occur in the second resolution table. In this manner, new levels of resolution tables can be added using additional bits selected from the hashed number until all entries are resolved. There is no need to resort to additional circuitry or techniques to resolve hash collisions.

[0016] In accordance with one embodiment of the invention, each hashed number is stored in a hash remainder table at an address indexed by the same index which is generated to index the output table. When the index has been generated, it is used to retrieve the stored hash number which is compared to the original hash number. A match indicates that the retrieval mechanism has retrieved a valid index and the forwarding information is retrieved from the output table and returned. If no match is obtained, default forwarding information is returned.

BRIEF DESCRIPTION OF THE DRAWINGS

[0017] The above and further advantages of the invention may be better understood by referring to the following description in conjunction with the accompanying drawings and which:

[0018] Figure 1 is a block schematic diagram of a data packet switching network on which the inventive method and apparatus may run.

[0019] Figure 2 is a more detailed block schematic diagram of a data packet switch which diagram illustrates the flow of data packets through the switch.

[0020] Figures 3A and 3B are schematic diagrams illustrating the data format in the three lookup hash tables in the case of no collision and in the case of a collision, respectively.

[0021] Figure 4 is a schematic diagram illustrating a translation process conducted in accordance with the principles of the present invention when no hash collision occurs.

[0022] Figure 5 is a schematic diagram illustrating translation processing in accordance with th principles of the pr sent invention wh n a hash collision occurs.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

[0023] Figure 1 illustrates, in very generalized form, a data packet local area network involving four stations 100-108, respectively, in which a translation method and apparatus constructed in accordance with the principles of the present invention can run. It should be understood that, although the invention is described in connection with a data packet switch and address translation, the inventive method and apparatus can also be used to perform other translations. In such a network, each station 100-108 contains software which breaks'the data into packets and affixes an appropriate header indicating how each packet should be routed through the network. Each station also contains software which receives data packets and reassembles the packets into the completed data stream.

[0024] Stations 100 and 102 are connected to a data packet switch 110 and stations 106 and 108 are connected to switch 114. Switches 110 and 114 route packets between the stations 100-108 and control access to the network. For example, station 100 may send packets over media line 116 to media input port 118 of switch 110. Switch 110 may route data packets to a destination station, for example station 106, based on the information in each packet header. To do this packets are first switched by switch 110 onto output line 112 to switch 114. Switch 114 then routes the packets to station 106. Although only one output line 112 is shown, switch 110 may have several output lines.

[0025] Each data packet switch, for example switch 100, acts as a packet relay. The input side of the switch contains a first in-first out (FIFO) buffer queue connected to the media input port 118. Packets arrive from the incoming communication link 116 and are entered into one end of the buffer. The switching process involves examining each packet, in turn, at the other end of the buffer and determining from the routing codes in the packet header which output line should be used. The packet is then added to a FIFO buffer queue for that output line which holds packets waiting to be transmitted on the associated outgoing communication link 112.

[0026] It is important that the switching process operate efficiently because data packets can arrive at closely spaced intervals. Switch congestion occurs when either the input or output FIFO buffers fill to a predetermined level. If the congestion becomes severe enough the buffers may become completely full, in which case there is no space for incoming packets to be stored and the packets are lost. With many transmission protocols, the successful transmission of a data through the network requires successful delivery of all the packets formed from the original data and the loss of any one packet may requires the entire piece of data to be retransmitted. When a station repeatedly tries to resend the same data, the network becomes increasingly loaded as more network congestion occurs and, accordingly, the possibility that more packets will be lost increases. As a result, many packet-switched networks can operate efficiently only when there is no packet loss.

[0027] Figure 2 illustrates, in more detail, the construction of a switch, such as switch 100 or 114. The heart of the switch is a data moving engine (DME) which includes a receive FIFO 200, a data packet memory 206, and associated memory control 212, and an output FIFO 218. Reception of data packets is controlled by lookup engine 202 operating in conjunction with memory control 212 and tables 204. Output of data packets is controlled by transmit poller and encapsulator 216. The DME is designed to move data packets between the input and output ports, 201, 220 and a local packet memory 206 and provide lookup functions on received packets for filtering and bridge forwarding.

[0028] More specifically, the movement of data between the input port 201, the output port 220 and the packet memory 206 occurs autonomously. Received packets are demultiplexed (if necessary) and put onto queues stored in the packet memory 206. These queues take the form of linked lists in the data packet memory 206. Packets waiting on queues designated as transmit queues are transmitted to the output port as the interfaces become available. To this end, the head of each output queue linked list is associated with an output port. Packets may also be received on non-transmit queues; these packets will require further processing by a data processor (not shown) and may include spanning tree packets, packets to be routed, packets requiring format translation. The processor places these latter packets on the transmit queues once they are processed, or else removes them for further local processing.

[0029] The receive FIFO 200 provides data buffering between the packet memory 206 and the input ports. One physical FIFO is provided which is logically partitioned into multiple FIFOs of which one FIFO is associated with each input port. The transmit FIFO 220 provides data buffering between the packet memory 206 and the output ports. As with the receive FIFO 200, only one physical FIFO is provided which is logically partitioned into multiple FIFOs of which one FIFO is associated with each output port.

[0030] After a packet has been received in the input port, the Lookup Engine 202 receives the header information from the input port, hashes it and uses the result to access the Hash Tables 204 stored in memory, as will be described in detail below. The result of the Hash and Lookup processes is a "forwarding value". The forwarding value is used by the memory control 212 to determine whether to queue or filter the packet and the appropriate queue or (filter counter) to use. The lookup engine 202 also generates "packet control words" 210 which are written into the packet memory 206 ahead of the packet data 208 for each packet (a single data packet is illustrated in memory 206 for clarity.)

[0031] Transmission or forwarding of the stored data packets is controlled by transmit poller 214. Transmit poller 214 is a process which polls certain queues in order to find data packets that can be transmitted. When such a data packet is found, it is dequeued and provided to th encapsulator 216. Encapsulator 216 uses the packet control words stored

with the data to determine whether and how the outgoing packet must be encapsulated. For example, the packet may need some header bytes appended.

[0032] The inventive lookup process is performed by lookup engine 202 which is a process which runs in the processor of the associated switch. The lookup argument provided to the lookup engine by the input port may be a 48-bit MAC address, or other information, such as a 5-byte Protocol Identifier (PID), a 2-byte Protocol Type (PT) or a one-byte Destination Service Access Point (DSAP). The lookup engine 202 pads each lookup argument with a type code and, if necessary, zero-value bytes to make all lookup arguments 48-bits wide. The 48-bit width is merely illustrative and any other lookup argument width can also be used in accordance with the principles of the invention.

[0033] More particularly, the lookup engine classifies each input value into one of four types. Padding is applied as necessary to make the argument 48 bits wide, and a 4-bit Type code is tagged to the argument. The following Table 1 illustrates type codes and padding values used for the various lookup argument types in accordance with one illustrative embodiment:

TABLE 1

Lookup Type	Type Code	Padding Prefix
MAC address	1	(none)
PID	8	8D
PT	9	9D 00 00 00
DSAP	A	AD 00 00 00 00

[0034] The padding values are added to the left of the argument string so that the least significant bit of the padded result (bit 0) corresponds to bit 0 of the rightmost byte in the lookup argument and the most significant bit of the result (bit 47) corresponds to the most significant bit of the leftmost byte of the (padded) result. In accordance with a preferred embodiment, the specific padding prefixes are chosen to minimize the chance of collisions with assigned IEEE multicast addresses. In addition, as described below, the lookup type is stored for a later comparison with the information retrieved by the hash lookup process. Therefore, there is no possibility that addresses can be confused with protocols. The lookup argument is then hashed to provide a hashed argument, which is used to access tables 204 to perform the inventive lookup process.

[0035] Tables 204 are sized and built at system initialization or later during operation whenever the network is reconfigured by adding stations or switches. They consist of three hash tables, and a hash remainder table. At system initialization time, the number and distribution of initial system addresses are known sot hat the hash and remainder tables can be constructed. Also at this time. a unique forwarding value is selected for each address in the address set which forwarding value will be used to route the packet through the switch. these forwarding values are stored in a forwarding table which is indexed by a forwarding index. Later when stations or ports are added or deleted. the corresponding addresses are added or deleted and tables 204 are recomputed.

[0036] As previously mentioned, the goal is to provide a reduction in the number of potential addresses represented by a 48-bit data packet address field to a much smaller number of addresses, typically represented by a 12 to 16 bit address field. In accordance with a preferred embodiment, the reduced address size, and, accordingly, the size of the hash tables can be selected programmatically. Theoretically, it would be possible to reduce the size of the 48-bit address by simply using a subset of the bits, for example, the least significant 12-16 bits. However, practically, the addresses often contain sequences in certain unpredictable bit positions that make address compression by simple bit field extraction unreliable.

[0037] Therefore, the lookup tables are built with a programmable randomizing function, called a hash function, designed to generate a small number (less than 64K) of indices corresponding to a set of (up to 64K) lookup arguments. Ideally, the mapping from arguments to indices is one-to-one, but a small number of many-to-one mappings can be tolerated. The hash function converts the 48-bit input argument into a new 48-bit output argument, called the hashed argument, using a process based on polynomial multiplication and division. Specifically, the bits of the input argument (either 1 or 0) are considered to represent the coefficients of an order-47 polynomial in x; if this is called A(x), then:

$$A(x) = a_{47} .x^{47} + a_{46} .x^{46} + ... + a_2 .x^2 + a_1 .x + a_0$$

where the coefficients a_0 - a_{47} are the bit values of the 48-bit input argument. In accordance with the hash function, this input polynomial is multipli d by a programmable hash multiplier which is stored in memory. The hash multiplier M(x) is represented by th polynomial:

$$M(x) = m_{47} .x^{47} + m_{46} .x^{46} + ... + m_2 .x^2 + m_0$$

[0038] The multiplication of the polynomials is done using modulo-2 addition; this yields an order-94 polynomial with coefficients that are also 0 or 1. This product is finally divided by a generator polynomial G(x), of fixed value given by:

$$G(x) = x^{48} + x^{36} + x^{25} + x^{10} + 1$$

[0039] The division is again carried out using modulo-2 addition. The division yields a quotient Q(x), which is a polynomial of order 46, and a remainder R(x) which is a polynomial of order 47. The operands are related by the equation:

$$A(x) * M(x) = Q(x) + R(x) * G(x)$$

[0040] The generator polynomial is chosen such that for any given multiplier polynomial (except 0), there is one-to-one mapping between the set of all values of A(x) and the corresponding set of all values R(x). The quotient Q(x) can therefore be discarded, as any value A(x) is derivable from its corresponding R(x). By choosing a suitable value for M (x), any given bounded set of lookup arguments (a subset of all values of A(x)) can be mapped onto a set of hashed arguments (a subset of all values of R(x)) with high likelihood of uniqueness in the low order 16 bits (maybe as few as 12 bits) of R(x). Statistical trials have shown that a random value M(x) is often sufficient, with retries beyond two attempts being very unlikely.

[0041] Since the multiplier polynomial M(x) can be programmatically selected, it can be changed and the hash table rebuilt during operation of the system if it is found that a large number of hash collisions is occurring with a particular set of addresses. This avoids problems which are often encountered with fixed hash functions and certain address sets. In order to improve performance, the multiplier polynomial should be selected to minimize the number of collisions and the hash table size should be selected so that it is in sparsely populated (for example, the hash table size may be on the order of four times larger than the number of address entries).

[0042] A first hash table is then constructed by first selecting the number of bits in the reduced address field which determines the size of the first hash table and the width of the first hash table words (each word has a width equal to the number of bits in the hash index plus one for a collision flag.) The first hash table is next initialized with a zero in each entry. Each address is then hashed in accordance with the hash function described above to generate a hashed argument. The hashed argument is then used as an index into the first hash table.

[0043] If the indexed location is empty, the zero entry is replaced by a data entry shown in Figure 3A. This entry has the most significant bit 300 assigned as a "collision flag" which is set to "0". The remaining bits are assigned to the unique forwarding index which is selected as previously described. In this case the complete hashed argument and the type code are also entered in the hash remainder table at a location indexed by the unique forwarding index.

[0044] Alternatively, if a forwarding index is already present in the indexed location, the entry illustrated in Figure 3B is made at the indexed location. In this entry, the collision flag 304 is set to "1" and the stored forwarding value is replaced by a collision entry. The collision entry comprises a bit mask field 306 and a next table index 308. The collision entry will instruct the lookup engine 202 to use more bits from the hashed argument, in conjunction with resolution areas in the additional hash tables 2 or 3 to resolve the collision.

[0045] The bit mask field is a 4-bit mask number which specifies one of sixteen functions which will be used to create a new 14-bit resolution table index from the 12-bit next table field and specified extra bits of the hashed argument when the lookup function is performed. This new resolution table index will be used to access a resolution table in either the second or third hash table to resolve the collision. The definition of the sixteen possible functions is shown in Table 2 below:

TABLE 2

Mask No.	Hash Table	Bit Mask	Shift Amount	Resolution Table Index
0000	2	00000011	2	NextTableIndex & "XX"
0001	2	00000101	3	NextTableIndex & "XX"
0010	2	00000110	3	NextTableIndex & "XX"
0011	2	00001001	4	NextTableIndex & "XX"
0100	2	00001010	4	NextTableIndex & "XX"

TABLE 2 (continued)

Mask N .	Hash Table	Bit Mask	Shift Am unt	Resolution Tabl Index
0101	2	00001100	4	NextTableIndex & "XX"
0110	2	00010001	5	NextTableIndex & "XX"
0111	2	00010010	5	NextTableIndex & "XX"
1000	2	00010100	5	NextTableIndex & "XX"
1001	2	00011000	5	NextTableIndex & "XX"
1010	3	00001111	4	NextTableIndex (11..2) & "XXXX"
1011	3	00000110	5	NextTableIndex (11..2) & "XXXX"
1100	3	00111100	6	NextTableIndex (11..2) & "XXXX"
1101	3	01111000	7	NextTableIndex (11..2) & "XXXX"
1110	3	11110000	8	NextTableIndex (11..2) & "XXXX"
1111	3	11111111	8	NextTableIndex (11..6) &
"XXXXXXXX"				

[0046] In this table, the values in the column titled "Hash Table" indicates whether the resolution table used to resolve the collision is part of the second hash table or part of the third hash table. The values in the column titled "Bit Mask" is an abstract mask that indicates which bits are to be selected from the low-order 8 bits of the *shifted* hashed argument. The "Shift Amount" column indicates how many bit positions to shift to the right the hashed argument after the bit mask is applied and bits to be used in the present resolution stage are selected. The shifting operation causes new bits of the hashed argument to be used during each stage of the resolution process. Finally, the column entitled "Resolution Table Index" indicates how to form the index into the resolution table by concatenating (indicated by the "&" symbol) the Next Table Index bits from the collision entry with the bits selected from the shifted hash argument (indicated by "X"s.)

[0047] In order to select the bit mask number, the set of addresses is examined to determine which of the bits in the upper 16 bits is most likely to change and a mask number is selected for which the corresponding bit mask selects those bits. This mask number is inserted in the mask number field of the collision entry. Then, a resolution table is created in the second or third hash tables in accordance with Table 2 above depending on the mask number selected. Resolution tables in the second hash table always have four entries and resolution tables in the third hash table always have either sixteen or 256 entries. The upper 12 bits of the resolution table base address are put into the Next Table Index Field of the collision entry. The mask is applied to the upper bits of the hashed address to determine the entry position in the newly created resolution table and entry as illustrated in Figure 3A is inserted into the table indicating the forwarding address.

[0048] Operation continues in this fashion. If a collision is detected in the second resolution table, then the forwarding address entry illustrated in Figure 3A is replaced by a collision entry as shown in Figure 3B and a new resolution table is created. The operation continues in this manner until all addresses have been processed.

[0049] Figure 4 illustrates the basic translation process which occurs when no collisions are detected. As previously mentioned there are four basic types of address information on which to perform a lookup operation. In the Figure, the address information is assumed to be 48 bits wide. This address information is loaded into register 400. A four bit type code is loaded into register 402 for a comparison as will be described in detail below.

[0050] The address information in register 400 and a multiplier M(x), discussed above and stored in register 406, are provided to a hasher process which generates a hashed argument in accordance with the technique as described above. The hashed argument is stored in register 408.

[0051] The lower 12 to 16 bits of the hashed argument in register 408 are used as a hash index to access the first hash table 420 as indicated by arrow 410. If the value zero (as initialized by the user and indicated, for example, by entry 422) is stored at the location identified by the hash index, then a forwarding index for the hashed argument is not stored in any of the hash tables. Conversely, if the first hash table entry is non-zero (for example, entries 424 and 426), then it is possible (though not definite) that a forwarding index is stored in one of the hash tables.

[0052] More specifically, if the hash index identifies a non-zero entry from the first hash table which indicates *no collision* (the collision flag is '0', as indicated by entry 426), then a possible forwarding index can be retrieved from the first hash table as illustrated by arrow 418. Since many addresses may alias to the same hash index, the forwarding index is used to access the hash remainder table, as indicated by arrow 434, and retrieve the type code 428 and hashed value of the address 430 previously stored at the location accessed by th retrieved forwarding index. Th retrieved type code is compared in comparator 412 to the typ code stored in register 402. The retrieved hashed argum nt is compared to th hashed argument stored in register 408 by comparator 416. the outputs of comparators

412 and 416 are provided to a decision process 414. If the hash remainder table entries 428 and 430 and the type code in register 402 and the hashed argument in register 408 are equal, the decision process 414 uses the retrieved forwarding index on line 434 to access a forwarding table 438 to retrieve a forwarding value. the forwarding value is used, as previously discussed, to route the data packet through the packet switch. Alternatively, if the comparators 412 and 416 do not both indicate a match, an alias has been found and a default index is returned.

[0053] If the hash index identifies a collision entry in the first hash table as illustrated by entry 424 in which the collision flag is "1", then further processing must be performed to resolve a collision caused when two or more different addresses generate the same hash index.

[0054] Figure 5 illustrates collision processing performed in accordance with the principles of the present invention. As shown in Figure 5, resolution tables in the second and third hash tables are used to resolve collisions until forwarding indexes are retrieved for all addresses. The resolution process proceeds in stages. At each stage in the collision resolution process additional bits from the hashed argument are used in combination with an index stored in the collision entry to access a resolution table. This process is repeated until a resolution is obtained.

[0055] More specifically, a hash index formed of the 12-16 least significant bits of the hashed argument 500 (stored in register 408 in Figure 4) is applied to the first hash table 504 as previously discussed. In the illustrative example shown in Figure 5, four addresses collide, or generate hashed arguments which index the same entry, in the first hash table 504. Accordingly, when the first hash table was generated as discussed above, a collision entry 506 has been made in the entry indexed by the hash index.

[0056] The collision entry 506 in the first hash table 504 contains a next table index (NTI) which points to a first resolution table 510 located in the second hash table address space. A mask field (indicated by 'M') in the collision entry 506 describes how to create an offset into the first resolution table 510 by using two bits from the next 2, 3, 4 or 5 higher order bits (31..16) of the hashed argument 500. As previously mentioned, the bit mask indicated which of the next 2-5 should be selected. The bits selected by the bit mask as indicated by arrow 509 are concatenated with the NTI as indicated by concatenator 508 to provide an index into the resolution table 510. After the index is generated the hashed argument in register 500 is shifted so that new bits of the hashed argument will be used in further stages of the resolution process.

[0057] As previously mentioned, all second hash table resolution tables have four entries, whose formats are identical to the first hash table entry formats illustrated in Figures 3A and 3B. Using the two bits selected by the bit mask M in the collision entry 506, in the illustrative example, three addresses still collide in the first resolution table 510. In particular, the three addresses index to entry 514. One address indexes to entry 512 and is accordingly, collision free. Therefore entry 512 contains a forwarding index. the translation process will check for a match in the hash remainder table using this forwarding index (if the address is accessed) as discussed above with respect to Figure 4.

[0058] The three colliding addresses require another resolution table 518 to be created. In the illustrative example, another resolution table located in the second hash table address space is used. Because there is a collision in the first resolution table, the table entry will have a collision entry 514 of the format illustrated in Figure 3B entered at the time the table is created. The bit mask in entry 514 specifies a further two bits from the next higher order 2 to 5 bits in the hashed argument that are to be used to index this second resolution table.

[0059] The selected bits as indicated by arrow 515, are concatenated by concatenator 516 with the next table index in entry 514 to access the second resolution table 518.

[0060] In this case, there are still two colliding addresses in the second resolution table 518 and one entry 520 is collision free. The collision free entry 520 is processes as described above in connection with the first resolution table. The entry 522 to which the two remaining addresses index is arranged to point to a third resolution table 526. This third resolution table 526 is located in the third hash table address space and contains either sixteen entries or 256 entries, which are indexed by 4 or 8 bits selected from a field of 4 to 8 higher order bits in the hashed argument. As with the previous resolution stages, the bit mask M in the collision entry 522 specifies which bits on the next least significant bits of the hashed argument are to be used to access the third resolution table 526. The selected bits as indicated by arrow 525 are concatenated by concatenator 524 and used to index the third resolution table 526.

[0061] In the illustrative example, the remaining two addresses are resolved in the third resolution table 526 as indicated by entries 530 and 532. Accordingly, each of these entries contains a forwarding index which is processed as previously described.

[0062] The amount of resolution processing which must be performed in accordance with the principles of the invention depends on the initial address set, the hashing method used and the size of the first hash table. Consequently, the values for the hash multiplier and the first hash table size should be selected so that collisions in the first hash table are minimized. It is also preferred to make the size of the hash table significantly larger (for example, four times larger) than the number of expected entries in order to spread the entries over the address space and minimize the possibility of collisions.

[0063] While the invention has been shown and described above with respect to various preferred embodiments, it will apparent that the foregoing and other changes of the form and detail may be made therein by one skilled in the art

without departing from the scope of the invention. These and other obvious modifications are intended to be covered by the following claims.

Claims

1. An apparatus for resolving a hash collision that occurs in a hash table (420) having a plurality of locations, the hash collision occurring when a first input number generates a first hashed value (408,500) and a second input number generates a second hashed value (408,500), wherein a first portion (410,502) of the first hashed value (408,500) and a first portion (410,502) of the second hashed value (408,500) have a same value, the first portion (410,502) of the first hashed value (408,500) being used to access a first location in the hash table, the apparatus **characterized by**:

   a first resolution table (510) containing a plurality of locations;
   a circuit to insert a collision entry (424,506) into the first location in the hash table (420,504) and into each hash table location where a hash collision occurs, the collision entry containing a bit mask number (M) identifying a masking function to select a second portion (509) of the first hashed value (408,500) and a pointer (NTI) to the first resolution table (508); and
   an index generator (508) to generate a first index, responsive to the collision entry (424,506) and to the second portion (509) of the first hashed value (408,500) selected by the masking function, to access a resolution entry (512,514) at a location in the first resolution table (510), the location being pointed to by the generated first index.

2. The apparatus according to claim 1, wherein the index generator (508) combines the pointer (NTI) contained in the collision entry (424,506) with the second portion (509) of the first hashed value (408,500). to access the resolution entry (512,514).

3. The apparatus according to claim 1, wherein the first portion (410,502) of the first hashed value (408,500) is used to access the first location in the hash table (420,504), and wherein the second portion (509) of the first hashed value (408,500) used by the index generator (508) is different from the first portion (410,502) of the first hashed value (408,500).

4. The apparatus of claim 1, wherein each of the plurality of locations of the hash table (420,504) containing a non-zero entry contains a flag set to one of a first logic value when the entry is a collision entry (424,506) and a second logic value when the entry is a non-collision entry (426).

5. The apparatus of claim 1, wherein the resolution entry is a non-collision entry (426) including a forwarding index (INDEX) to an output table (438) which contains a first output number (436).

6. The apparatus of claim 1, wherein the resolution entry (514) is a collision entry including a bit mask (M) number identifying a masking function and a pointer (NTI) to a second resolution table (518) with a plurality of locations.

7. The apparatus of claim 1, further comprising a hasher (404) to receive a plurality of input numbers, and operative to generate a hashed value (408,500) for each input number of the plurality of input numbers, each hashed value (408,500) used to access a location in the hash table (420,504).

8. The apparatus of claim 1, wherein each of the plurality of locations of the hash table (420,504) that contain a non-zero non-collision entry include a forwarding index to an output table (438) that contains a first output number (436).

9. The apparatus of claim 8, wherein the apparatus classifies the first input number as one of four types, the apparatus further comprising a hash remainder table (432) including a plurality of entries, a first entry containing a hashed value (430) and a type (428) of the first input number, the apparatus operative to verify a forwarding index (FOR IDX) of a hash table entry accessed by the first hashed value (408,500) using the first entry of the hash remainder table (432), which is pointed to (434) by the forwarding index (FOR IDX).

10. The apparatus of claim 1, wherein th masking function specifies bits to be selected from the first hashed value (408,500).

11. The apparatus of claim 1, wherein the masking function specifies a number of bit positions to shift the first hashed value (408,500) after the second portion (509) of the first hashed value (408,500) has been selected.

12. The apparatus of claim 1, wherein the masking function specifies how to combine the collision entry (424,506) with the second portion (509) of the first hashed value (408,500) to generate the first index.

13. A method of resolving a hash collision that occurs in a hash table (420) having a plurality of locations, the hash collision occurring when a first input number generates a first hashed value (408,500) and a second input number generates a second hashed value (408,500), wherein a first portion (410,503) of the first hashed value (408,500) and a first portion (410,502) of the second hashed value (408,500) have a same value, the first portion (410,502) of the first hashed value (408,500) being used to access a first location in the hash table, the method **characterized by**:

creating a resolution table (510) containing a plurality of entries;
inserting a collision entry into the first location in the hash table (504) and into each hash table location where a hash collision occurs, the collision entry (424,506) containing a bit mask number (M) identifying a masking function to select a second portion (509) of the first hashed value (408,500) and a pointer (NTI) to the resolution table (510);
generating an index responsive to the collision entry (506) and the second portion (509) of the first hashed value (408,500) selected by the masking function; and
accessing a resolution entry (512,514) at a location in the resolution table (510) pointed to by the generated index.

14. The method according to claim 13, wherein generating an index comprises:

combining the collision entry pointer (NTI) with the second portion (509) of the first hashed value (408,500).

15. The method according to claim 13, further comprising:

using the first portion (410,502) of the first hashed value (408,500) to access the first location in the hash table (420,504), the first portion (410,503) of the first hashed value (408,500) being different from the second portion (509) of the first hashed value (408,500).

16. The method of claim 13, further comprising inserting into each of the plurality of locations of the hash table (420,504) containing a non-zero entry a flag set to one of a first logic value when the entry is a collision entry (424,506) and a second logic value when the entry is a non-collision entry (426).

17. The method of claim 13, wherein the resolution entry is a non-collision entry (512), the method further comprising inserting into the resolution entry a forwarding index (INDEX) to an output table (438) that contains an output number (436).

18. The method of claim 13, wherein the resolution entry is a collision entry (514), the method further comprising inserting into the resolution entry a bit mask number (M) identifying a masking function and a pointer (NTI) to another resolution table (518) having a plurality of locations.

19. The method of claim 13, further comprising:

hashing a plurality of input numbers to generate a hashed value (408,500) for each of the input numbers; and
applying each hashed value (408,500) to access a location in the hash table (420,504).

20. The method of claim 13, further comprising inserting a forwarding entry index (FOR IDX) to an output table (438) that contains a first output number (436) into each of the plurality of locations of the hash table (420,504) that contain a non-zero collision entry.

21. The method of claim 20, further comprising:

classifying the first input number as one of four types;
generating a hash remainder table (432) having a plurality of entries, a first entry containing a hashed value

(430) and a type (428) of the first input number; and
verifying a forwarding index (FOR IDX) of a hash table entry accessed by the first hashed value (408,500) using the first entry of the hash remainder table (432), which is pointed to by the forwarding index (FOR IDX).

22. The method of claim 13, wherein the masking function specifies a number of bit positions to shift the first hashed value (408,500) after the second portion (509) of the first hashed value (408,500) has been selected.

23. The method of claim 13, wherein the masking function specifies how to combine the collision entry (424,506) with the second portion (509) of the first hashed value (424,506) to generate the index.

24. The method of claim 13, wherein the masking function specifies bits to be selected from the first hashed value (408,500).

25. A translation apparatus for translating a plurality of input numbers into a plurality of output numbers, the apparatus comprising:

a hasher (404), responsive to each of the plurality of input numbers, to generate a hashed argument (408,500) corresponding to the each input number, the hashed argument (408,500) having a plurality of bits;
a hash table (420,504) containing a plurality of entries, each entry of the plurality of entries containing either an index (FOR IDX) to an output table (438) which contains a first output number (436) or a bit mask number (M) identifying a masking function and a pointer (NTI) to a resolution table (518);
a first resolution table (510) containing a plurality of entries, each entry of the plurality of entries containing either an index (INDEX) into the output table (438) which contains a second output number or a bit mask number (M) identifying a masking function and a pointer (NTI) to a second resolution table (518);
a first index generator responsive to a first subset (410,502) of the plurality of bits of the hashed argument (408,500) to access the hash table (420,504); and
a second index generator (508) to access the first resolution table (510), responsive to an entry (506) in the hash table (420,504) that contains a bit mask number (M) and a pointer (NTI), and responsive to a second subset (509) of the plurality of bits of the hashed argument (408,500), wherein no bit from the hashed argument (408,500) is included in both the first subset (410,502) and second subset (509).

26. The apparatus of claim 25, wherein the apparatus classifies each of the plurality of input numbers as one of four types, the apparatus further comprising a hash remainder table (432) having a plurality of entries, each entry containing a hashed value (430) and a type (428) of the input number, the apparatus operative to verify an index of an entry of either the hash table (420,504) or the first resolution table (510) using an entry of the hash remainder table (432) pointed to by the index.

27. The apparatus of claim 25, wherein each location of the plurality of locations in each of the hash table (420,504) and the first resolution table (510) includes a collision flag, wherein for each entry of the plurality of entries where a hash collision does not occur the flag has a first logic value, and for each entry of the plurality of entries where a hash collision occurs the flag has a second logic value.

Patentansprüche

1. Vorrichtung zum Auflösen einer Hash-Kollision, die in einer Hash-Tabelle (420) mit einer Vielzahl von Speicherzellen vorkommt, die Hash-Kollision kommt vor wenn eine erste Eingabezahl einen ersten Hash-Wert (408, 500) erzeugt und eine zweite Eingabezahl einen zweiten Hash-Wert (408, 500) erzeugt, bei der ein erster Teil (410, 502) des ersten Hash-Wertes (408, 500) und ein erster Teil (410, 502) des zweiten Hash-Wertes (408, 500) einen gleichen Wert aufweisen, der erste Teil (410, 502) des ersten Hash-Wertes (408, 500) dazu genutzt wird, auf eine erste Speicherstelle in der Hash-Tabelle zuzugreifen, wobei die Vorrichtung **gekennzeichnet ist durch:**

eine erste Auflösungstabelle (510), die eine Vielzahl von Speicherstellen enthält;

eine Schaltung, um einen Eintrag einer Kollision (424, 506) in die erste Speicherstelle in der Hash-Tabell (420, 504) und in jede Hash-Tab llen-Sp icherstelle, bei der eine Hash-Kollision vorkommt, einzufügen, wobei der Eintrag einer Kollision eine Bitausblendzahl (M) enthält, die eine Ausblendfunktion identifiziert, um einen zweiten Teil (509) des rsten Hash-Wertes (408, 500) und einen Zeiger (NTI) auf di erst Auflösungstabelle

(508) auszuwählen;

einen Indexgenerator (508), um einen ersten Index, der auf den Eintrag einer Kollision (424, 506) und auf den zweiten Teil (509) des ersten Hash-Wertes (408, 500), der **durch** die Ausblendfunktion ausgewählt ist, ansprechempfindlich ist, zu erzeugen, um auf einen Eintrag einer Auflösung (512, 514) bei einer Speicherstelle in der ersten Auflösungstabelle (510) zuzugreifen, wobei auf die Speicherstelle **durch** den ersten erzeugten Index gezeigt wird.

2. Vorrichtung gemäß Anspruch 1,
bei der der Indexgenerator (508) den Zeiger (NTI), der in dem Eintrag einer Kollision (424, 506) mit dem zweiten Teil (509) des ersten Hash-Wertes (408, 500) enthalten ist, verbindet, um auf den Eintrag einer Auflösung (512, 514) zuzugreifen.

3. Vorrichtung gemäß Anspruch 1,
bei der der erste Teil (410, 502) des ersten Hash-Wertes (408, 500) verwendet wird, um auf den ersten Speicherplatz in der Hash-Tabelle (420, 504) zuzugreifen, und bei der der zweite Teil (509) des ersten Hash-Wertes (408, 500), der von dem Indexgenerator (508) genutzt wird, von dem ersten Teil (410, 502) des ersten Hash-Wertes (408, 500) verschieden ist.

4. Vorrichtung gemäß Anspruch 1,
bei der jede der Vielzahl von Speicherstellen der Hash-Tabelle (420, 504), die einen von null unterschiedlichen Eintrag enthält, einen Flag-Satz entweder eines ersten logischen Wertes, wenn der Eintrag ein Eintrag einer Kollision (424, 506) ist, oder eines zweiten logischen Wertes, wenn der Eintrag ein Eintrag einer Nicht-Kollision (426) ist, enthält.

5. Vorrichtung nach Anspruch 1,
bei der der Eintrag einer Auflösung ein Eintrag einer Nicht-Kollision (426) ist, die einen Weiterleitungsindex (INDEX) auf eine Ausgabetabelle (438), die eine erste Ausgabezahl (436) enthält, einschließt.

6. Vorrichtung nach Anspruch 1,
bei der der Eintrag einer Auflösung (514) ein Eintrag einer Kollision ist, die eine Bitausblendzahl (M), die eine Ausblendfunktion und einen Zeiger (NTI) auf eine zweite Auflösungstabelle (518) mit einer Vielzahl von Speicherstellen erkennt, einschließt.

7. Vorrichtung nach Anspruch 1,
die ferner einen Hasher (404) zum Empfangen einer Vielzahl von Eingabezahlen und im Betrieb zum Erzeugen eines Hash-Wertes (408, 500) für jede Eingabezahl der Vielzahl von Eingabezahlen umfasst, wobei jeder Hash-Wert (408, 500) verwendet wird, um auf eine Speicherstelle in der Hash-Tabelle (420, 504) zuzugreifen.

8. Vorrichtung gemäß Anspruch 1,
bei der jede der Vielzahl von Speicherstellen der Hash-Tabelle (420, 504), die einen von null verschiedenen Eintrag einer Nicht-Kollision enthalten, einen Weiterleitungsindex auf eine Ausgabetabelle (438), die eine erste Ausgabezahl (436) enthält, einschließt.

9. Vorrichtung gemäß Anspruch 8,
bei dem die Vorrichtung die erste Eingabezahl als eine von vier Typen klassifiziert,
die Vorrichtung ferner eine Hash-Rückstandstabelle (432) enthält, die eine Vielzahl von Einträgen umfaßt, wobei ein erster Eintrag einen Hash-Wert (430) und einen Typ (428) der ersten Eingabezahl enthält,
die Vorrichtung im Betrieb einen Weiterleitungsindex (FOR IDX) des Eintrags der Hash-Tabelle, auf den durch den ersten Hash-Wert (408, 500) durch Benutzung des ersten Eintrags der Hash-Rückstandstabelle (432), auf die durch den Weiterleitungsindex (FOR IDX) gezeigt wird (434), zugegriffen wird, verifiziert.

10. Vorrichtung gemäß Anspruch 1,
bei der die Ausblendfunktion Bits spezifiziert, die von dem ersten Hash-Wert (408, 500) ausgewählt werden sollen.

11. Vorrichtung gemäß Anspruch 1,
bei der die Ausblendfunktion eine Anzahl von Bitpositionen spezifiziert, um den ersten Hash-Wert (408, 500), nachdem der zweite Teil (509) des ersten Hash-Wertes (408, 500) ausgewählt worden ist, zu verschieben.

12. Vorrichtung gemäß Anspruch 1,
bei der die Ausblendfunktion spezifiziert, wie der Eintrag einer Kollision (424, 506) mit dem zweiten Teil (509) des ersten Hash-Wertes (408, 500) zu verbinden ist, um den ersten Index zu erzeugen.

13. Verfahren zum Auflösen einer Hash-Kollision, die in einer Hash-Tabelle (420) mit einer Vielzahl von Speicherzellen vorkommt, die Hash-Kollision kommt vor wenn eine erste Eingabezahl einen ersten Hash-Wert (408, 500) erzeugt und eine zweite Eingabezahl einen zweiten Hash-Wert (408, 500) erzeugt, worin ein erster Teil (410, 502) des ersten Hash-Wertes (408, 500) und ein erster Teil (410, 502) des zweiten Hash-Wertes (408, 500) einen gleichen Wert aufweisen, der erste Teil (410, 502) des ersten Hash-Wertes (408, 500) dazu genutzt wird, auf eine erste Speicherstelle in der Hash-Tabelle zuzugreifen, wobei das Verfahren **gekennzeichnet ist durch**:

Erschaffen einer Auflösungstabelle (510), die eine Vielzahl von Einträgen enthält;

Einfügen eines Eintrages einer Kollision in die erste Speicherstelle in der Hash-Tabelle (504) und in jede Hash-Tabellen-Speicherstelle, bei der eine Hash-Kollision vorkommt, einzufügen, wobei der Eintrag einer Kollision (424, 506) eine Bitausblendzahl (M) enthält, die eine Ausblendfunktion identifiziert, um einen zweiten Teil (509) des ersten Hash-Wertes (408, 500) und einen Zeiger (NTI) auf die Auflösungstabelle (510) auszuwählen;

Erzeugen eines Index, der auf den Eintrag einer Kollision (506) und auf den zweiten Teil (509) des ersten Hash-Wertes (408, 500), der **durch** die Ausblendfunktion ausgewählt ist, ansprechempfindlich ist; und

Zugreifen auf einen Eintrag einer Auflösung (512, 514) bei einer Speicherstelle in der ersten Auflösungstabelle (510), auf die **durch** den erzeugten Index gezeigt wird.

14. Verfahren gemäß Anspruch 13,
bei dem das Erzeugen eines Indexes umfasst:

Verbinden des Zeigers (NTI) eines Kollisioneintrages mit dem zweiten Teil (509) des ersten Hash-Wertes (408, 500).

15. Verfahren gemäß Anspruch 13,
welches ferner umfasst:

Verwenden des ersten Teils (410, 502) des ersten Hash-Wertes (408, 500), um auf den ersten Speicherplatz in der Hash-Tabelle (420, 504) zuzugreifen, wobei der erste Teil (410, 503) des ersten Hash-Wertes (408, 500) von dem zweiten Teil (509) des ersten Hash-Wertes (408, 500) verschieden ist.

16. Verfahren gemäß Anspruch 13,
welches ferner umfasst,
Einfügen einen Flag-Satz entweder eines ersten logischen Wertes, wenn der Eintrag ein Eintrag einer Kollision (424, 506) ist, und eines zweiten logischen Wertes, wenn der Eintrag ein Eintrag einer Nicht-Kollision (426) ist, in jede der Vielzahl von Speicherstellen der Hash-Tabelle (420, 504), die einen von null unterschiedlichen Eintrag enthält.

17. Verfahren gemäß Anspruch 13,
bei dem der Eintrag einer Auflösung ein Eintrag einer Nicht-Kollision (512) ist, das Verfahren umfasst ferner,
Einfügen eines Weiterleitungsindex (INDEX) auf eine Ausgabetabelle (438), die eine Ausgabezahl (436) enthält, in den Eintrag einer Auflösung.

18. Verfahren gemäß Anspruch 13,
bei dem der Eintrag einer Auflösung ein Eintrag einer Kollision (514) ist, das Verfahren umfasst ferner,
Einfügen einer Bitausblendzahl (M), die eine Ausblendfunktion und einen Zeiger (NTI) auf eine weitere Auflösungstabelle (518) mit einer Vielzahl von Speicherstellen identifiziert, in den Eintrag einer Auflösung.

19. Verfahren gemäß Anspruch 13,
welches ferner umfasst:

Hachen einer Vielzahl von Eingabezahlen, um einen Hash-Wert (408, 500) für jede Eingabezahl zu erzeugen;

und
Verwenden eines jeden Hash-Wertes (408, 500), um auf eine Speicherstelle in der Hash-Tabelle (420, 504) zuzugreifen.

20. Verfahren gemäß Anspruch 13,
welches ferner umfasst,
Einfügen eines Weiterleitungseintragsindex (FOR IDX) auf eine Ausgabetabelle (438), die eine erste Ausgabezahl (436) in jede der Vielzahl von Speicherzellen der Hash-Tabelle (420, 504), die einen von null verschiedenen Eintrag einer Kollision enthalten, enthält.

21. Verfahren gemäß Anspruch 20,
welches ferner umfasst:

Klasifizieren der ersten Eingabezahl als eine von vier Typen;
Erzeugen einer Hash-Rückstandstabelle (432) mit einer Vielzahl von Einträgen, wobei ein erster Eintrag einen Hash-Wert (430) und einen Typ (428) der ersten Eingabezahl enthält; und
Verifizieren eines Weiterleitungsindex (FOR IDX) eines Eintrags der Hash-Tabelle, auf den von dem ersten Hash-Wert (408, 500) durch Benutzung des ersten Eintrags der Hash-Rückstandstabelle (432), auf die durch den Weiterleitungsindex (FOR IDX) gezeigt wird, zugegriffen wird.

22. Verfahren gemäß Anspruch 13,
bei dem die Ausblendfunktion eine Anzahl von Bitpositionen spezifiziert, um den ersten Hash-Wert (408, 500), nachdem der zweite Teil (509) des ersten Hash-Wertes (408, 500) ausgewählt worden ist, zu verschieben.

23. Verfahren gemäß Anspruch 13,
bei dem die Ausblendfunktion spezifiziert, wie der Eintrag einer Kollision (424, 506) mit dem zweiten Teil (509) des ersten Hash-Wertes (408, 500) zu verbinden ist, um den ersten Index zu erzeugen.

24. Verfahren gemäß Anspruch 13,
bei dem die Ausblendfunktion Bits spezifiziert, die von dem ersten Hash-Wert (408, 500) ausgewählt werden sollen.

25. Übersetzungsvorrichtung zum Übersetzen einer Vielzahl von Eingabezahlen in eine Vielzahl von Ausgabezahlen, bei der die Vorrichtung umfasst:

einen Hasher (404), der auf jede der Vielzahl von Eingabezahlen ansprechempfindlich ist, um ein Hash-Argument (408, 500) entsprechend jeder Eingabezahl zu erzeugen, wobei das Hash-Argument (408, 500) eine Vielzahl von Bits aufweist;

eine Hash-Tabelle (420, 504), die eine Vielzahl von Einträgen enthält, wobei jeder Eintrag der Vielzahl von Einträgen entweder einen Index (FOR IDX) auf eine Ausgabetabelle (438), die eine erste Ausgabezahl (436) enthält, oder eine Bitausblendzahl (M), die eine Ausblendfunktion und einen Zeiger (NTI) auf einer zweiten Auflösungstabelle (518) identifiziert, enthält;

eine erste Auflösungstabelle (510), die eine Vielzahl von Einträgen aufweist, wobei jeder Eintrag der Vielzahl von Einträgen entweder einen Index (INDEX) in die Ausgabetabelle (438), der eine zweite Ausgabezahl enthält, oder eine Bitausblendzahl (M), die eine Ausblendfunktion und einen Zeiger (NTI) auf eine zweite Auflösungstabelle (518) identifiziert, enthält;

einen ersten Indexgenerator, der, um auf die Hash-Tabelle (420, 504) zuzugreifen, ansprechempfindlich ist auf eine erste Teilmenge (410, 502) der Vielzahl von Bits des Hash-Arguments (408, 500); und

einen zweiten Indexgenerator (508), um auf die erste Auflösungstabelle (510) zuzugreifen, die ansprechempfindlich auf einen Eintrag (506) in der Hash-Tabelle (420, 504), die eine Bitausblendzahl (M) und einen Zeiger (NTI) enthält, ist, und die auf eine zweite Teilmenge (509) der Vielzahl von Bits des Hash-Arguments (408, 500) ansprechempfindlich ist, wobei kein Bit von dem Hash-Arugument (408, 500) sowohl in die erste Teilmenge (410, 502) und in die zweite Teilmenge (509) eingeschlossen wird.

26. Vorrichtung gemäß Anspruch 25,

wobei die Vorrichtung jede Vielzahl von Eingabezahlen als eine von vier Typen klassifiziert.

27. Vorrichtung gemäß Anspruch 25,
wobei jede Speicherstelle der Vielzahl von Speicherstellen in jeder Hash-Tabelle (420, 504) und ersten Auflösungstabelle (510) einen Kollisions-Flag einschließt, wobei für jeden Eintrag der Vielzahl von Einträgen, bei denen eine Hash-Kollision nicht auftritt, das Flag einen ersten logischen Wert aufw ist, und bei dem für jeden Eintrag der Vielzahl von Einträgen, bei denen eine Hash-Kollision auftritt, das Flag einen zweiten logischen Wert aufweist.

Revendications

1. Appareil destiné à résoudre une collision de hachage se produisant dans une table de hachage (420) ayant une pluralité d'emplacements, la collision de hachage se produisant lorsqu'un premier nombre d'entrée génère une première valeur hachée (408, 500) et lorsqu'un second nombre d'entrée génère une seconde valeur hachée (408, 500), dans lequel une première partie (410, 502) de la première valeur hachée (408, 500) et une première partie (410, 502) de la seconde valeur hachée (408, 500) ont la même valeur, la première partie (408, 502) de la première valeur hachée (408, 500) étant utilisée pour accéder à un premier emplacement dans la table de hachage, l'appareil étant **caractérisé par** :

   une première table de résolution (510) contenant une pluralité d'emplacements ;
   un circuit pour insérer une entrée de collision (424, 506) au premier emplacement de la table de hachage (420, 504) et à chaque emplacement de la table de hachage où il se produit une collision de hachage, l'entrée de collision contenant un nombre de masquage de bits (M) identifiant une fonction de masquage pour sélectionner une seconde partie (509) de la première valeur hachée (408, 500) et un pointeur (NTI) sur la première table de résolution (508) ; et
   un générateur d'index (508) pour générer un premier index en réponse à l'entrée de collision (424, 506) et à la seconde partie (509) de la première valeur hachée (408, 500) sélectionné par la fonction de masquage, pour accéder à une entrée de résolution (512, 514) à un emplacement de la première table de résolution (510), le premier index généré pointant sur cet emplacement.

2. Appareil selon la revendication 1, dans lequel le générateur d'index (508) combine le pointeur (NTI) contenu dans l'entrée de collision (424, 506) avec la seconde partie (509) de la première valeur hachée (408, 500) pour accéder à l'entrée de résolution (512, 514).

3. Appareil selon la revendication 1, dans lequel la première partie (410, 502) de la première valeur hachée (408, 500) est utilisée pour accéder au premier emplacement de la table de hachage (420, 504), et dans lequel la seconde partie (509) de la première valeur hachée (408, 500) utilisée par le générateur d'index (508) est différente de la première partie (410, 502) de la première valeur hachée (408, 500).

4. Appareil selon la revendication 1, dans lequel chacun de la pluralité d'emplacements de la table de hachage (420, 504) contenant une entrée non nulle, contient un indicateur positionné à l'une d'une première valeur logique lorsque l'entrée est une entrée de collision (424, 506) et d'une seconde valeur logique lorsque l'entrée est une entrée de non collision (426).

5. Appareil selon la revendication 1, dans lequel l'entrée de résolution est une entrée de non collision (426) comportant un index de redirection (INDEX) vers une table de sortie (438) qui contient un premier nombre de sortie (436).

6. Appareil selon la revendication 1, dans lequel l'entrée de résolution (514) est une entrée de collision comportant un nombre de masquage de bits (M) identifiant une fonction de masquage et un pointeur (NTI) sur une seconde table de résolution (518) ayant une pluralité d'emplacements.

7. Appareil selon la revendication 1, comprenant en outre un hacheur (404) destiné à recevoir une pluralité de nombres d'entrée et ayant pour fonction de générer une valeur hachée (408, 500) pour chaque nombre d'entrée de la pluralité de nombres d'entrée, chaque valeur hachée (408, 500) étant utilisée pour accéder à un emplacement de la table de hachage (420, 504).

8. Appareil selon la revendication 1, dans lequel chacun de la pluralité d' mplacements de la table de hachag (420, 504) qui contiennent une entré de non collision non nulle comport un index de redirection vers une table de

sortie (438) qui contient un premier nombre de sortie (436).

9. Appareil selon la revendication 8, dans lequel l'appareil classe le premier nombre d'entrée comme étant l'un de quatre types, l'appareil comprenant en outre une table de restes de hachage (432) comportant une pluralité d'entrées, une première entrée contenant une valeur hachée (430) et un type (428) du premier nombre d'entrée, l'appareil ayant pour fonction de vérifier un index de redirection (FOR IDX) d'une entrée de la table de hachage à laquelle accède la première valeur hachée (408, 500) en utilisant la première entrée de la table de restes de hachage (432), sur laquelle pointe (434) l'index de redirection (FOR IDX).

10. Appareil selon la revendication 1, dans lequel la fonction de masquage spécifie des bits devant être sélectionnés à partir de la première valeur hachée (408, 500).

11. Appareil selon la revendication 1, dans lequel la fonction de masquage spécifie un nombre de positions de bits dont la première valeur hachée (408, 500) doit être décalée après que la seconde partie (509) de la première valeur hachée (408, 500) a été sélectionnée.

12. Appareil selon la revendication 1, dans lequel la fonction de masquage spécifie la façon dont on combine l'entrée de collision (424, 506) avec la seconde partie (509) de la première valeur hachée (408, 500) pour générer le premier index.

13. Procédé de résolution d'une collision de hachage se produisant dans une table de hachage (420) ayant une pluralité d'emplacements, la collision de hachage se produisant lorsqu'un premier nombre d'entrée génère une première valeur hachée (408, 500) et lorsqu'un second nombre d'entrée génère une seconde valeur hachée (408, 500), dans lequel une première partie (410, 503) de la première valeur hachée (408, 500) et une première partie (410, 502) de la seconde valeur hachée (408, 500) ont la même valeur, la première partie (410, 502) de la première valeur hachée (408, 500) étant utilisée pour accéder à un premier emplacement de la table de hachage, le procédé étant **caractérisé par** :

la création d'une table de résolution (510) contenant une pluralité d'entrées ;
l'insertion d'une entrée de collision au premier emplacement de la table de hachage (504) et à chaque emplacement de la table de hachage où il se produit une collision de hachage, l'entrée de collision (424, 506) contenant un nombre de masquage de bits (M) identifiant une fonction de masquage pour sélectionner une seconde partie (509) de la première valeur hachée (408, 500) et un pointeur (NTI) sur la première table de résolution (510) ;
la génération d'un index en réponse à l'entrée de collision (506) et à la seconde partie (509) de la première valeur hachée (408, 500) sélectionnée par la fonction de masquage ; et
l'accès à une entrée de résolution (512, 514) à un emplacement de la table de résolution (510) sur lequel pointe l'index généré.

14. Procédé selon la revendication 13, dans lequel la génération d'un index comprend :

la combinaison du pointeur d'entrée de collision (NTI) avec la seconde partie (509) de la première valeur hachée (408, 500).

15. Procédé selon la revendication 13, comprenant en outre :

l'utilisation de la première partie (410, 502) de la première valeur hachée (408, 500) pour accéder au premier emplacement de la table de hachage (420, 504), la première partie (410, 503) de la première valeur hachée (408, 500) étant différente de la seconde partie (509) de la première valeur hachée (408, 500).

16. Procédé selon la revendication 13, comprenant en outre l'insertion dans chacun de la pluralité d'emplacements de la table de hachage (420, 504) contenant une entrée non nulle, un indicateur positionné à l'une d'une première valeur logique lorsque l'entrée est une entrée de collision (424, 506) et d'une seconde valeur logique lorsque l'entrée est une entrée de non collision (426).

17. Procédé selon la revendication 13, dans lequel l'entrée de résolution est une entrée de non collision (512), le procédé comprenant en outre l'insertion dans l'entrée de résolution d'un index de redirection (INDEX) vers une table de sortie (438) qui contient un nombre de sortie (436).

18. Procédé selon la revendication 13, dans lequel l'entrée de résolution est une entrée de collision (514), le procédé comprenant en outre l'insertion dans l'entrée de résolution d'un nombre de masquage de bits (M) identifiant une fonction de masquage et d'un pointeur (NTI) sur une autre table de résolution (518) ayant une pluralité d'emplacements.

19. Procédé selon la revendication 13, comprenant en outre :

le hachage d'une pluralité de nombres d'entrée pour produire une valeur hachée (408, 500) pour chacun des nombres d'entrée ; et
l'application de chaque valeur hachée (408, 500) pour accéder à un emplacement de la table de hachage (420, 504).

20. Procédé selon la revendication 13 comprenant en outre l'insertion d'un index d'entrée de redirection (FOR IDX) vers une table de sortie (438) qui contient un premier nombre de sortie (436) dans chacun de la pluralité d'emplacements de la table de hachage (420, 504) qui contiennent une entrée de collision non nulle.

21. Procédé selon la revendication 20, comprenant en outre :

le classement du premier nombre d'entrée comme étant de l'un de quatre types ;
la génération d'une table de restes de hachage (432) comportant une pluralité d'entrées, une première entrée contenant une valeur hachée (430) et un type (428) du premier nombre d'entrée ; et
la vérification d'un index de redirection (FOR IDX) d'une entrée de la table de hachage à laquelle accède la première valeur hachée (408, 500) en utilisant la première entrée de la table de restes de hachage (432), sur laquelle pointe l'index de redirection (FOR IDX).

22. Procédé selon la revendication 13, dans lequel la fonction de masquage spécifie un nombre de positions de bits dont la première valeur hachée (408, 500) doit être décalée après que la seconde partie (509) de la première valeur hachée (408, 500) a été sélectionnée.

23. Procédé selon la revendication 13, dans lequel la fonction de masquage spécifie la façon dont on combine l'entrée de collision (424, 506) avec la seconde partie (509) de la première valeur hachée (424, 506) pour générer l'index.

24. Procédé selon la revendication 13, dans lequel la fonction de masquage spécifie les bits devant être sélectionnés à partir de la première valeur hachée (408, 500).

25. Appareil de traduction pour traduire une pluralité de nombres d'entrée en une pluralité de nombres de sortie, l'appareil comprenant :

un hacheur (404), répondant à chacun de la pluralité de nombres d'entrée en générant un argument haché (408, 500) correspondant à chaque nombre d'entrée, l'argument haché (408, 500) ayant une pluralité de bits ;
une table de hachage (420, 504) contenant une pluralité d'entrées, chaque entrée de la pluralité d'entrées contenant soit un index (FOR IDX) sur une table de sortie (438) qui contient un premier nombre de sortie (436), soit un nombre de masquage de bits (M) identifiant une fonction de masquage et un pointeur (NTI) sur une table de résolution (518) ;
une première table de résolution (510) contenant une pluralité d'entrées, chaque entrée de la pluralité d'entrées contenant soit un index (INDEX) dans la table de sortie (438) qui contient un second nombre de sortie, soit un nombre de masquage de bits (M) identifiant une fonction de masquage et un pointeur (NTI) sur une seconde table de résolution (518) ;
un premier générateur d'index répondant à un premier sous-ensemble (410, 502) de la pluralité de bits de l'argument haché (408, 500) en accédant à la table de hachage (420, 504) ; et
un second générateur d'index (508) pour accéder à la première table de résolution (510), en réponse à une entrée (506) de la table de hachage (420, 504) qui contient un nombre de masquage de bits (M) et un pointeur (NTI), et répondant à un second sous-ensemble (509) de la pluralité de bits de l'argument haché (408, 500), dans lequel aucun bit provenant de l'argument haché (408, 500) n'est contenu ni dans le premier sous-ensemble (410, 502), ni dans le second sous-ensembl (509).

26. Appareil s lon la r vendication 25, dans lequel l'appareil classe chacun de la pluralité de nombr s d'entrée comme étant de l'un de quatre types, l'appareil comprenant en outre une table de restes de hachage (432) ayant une

pluralité d'entrées, chaque entrée contenant une valeur hachée (430) et un type (428) du nombre d'entrée, l'appareil ayant pour fonction de vérifier un index d'une entrée soit de la table de hachage (420, 504), soit de la première table de résolution (510) en utilisant une entrée de la table de restes de hachage (432) sur laquelle pointe l'index.

27. Appareil selon la revendication 25, dans lequel chaque emplacement de la pluralité d'emplacements de chaque table de hachage (420, 504) et de la première table de résolution (510) contient un indicateur de collision, dans lequel, pour chaque entrée de la pluralité d'entrées pour lesquelles il ne se produit pas de collision de hachage, l'indicateur a une première valeur logique, et pour chaque entrée de la pluralité d'entrées où il se produit une collision de hachage, l'indicateur a une seconde valeur logique.

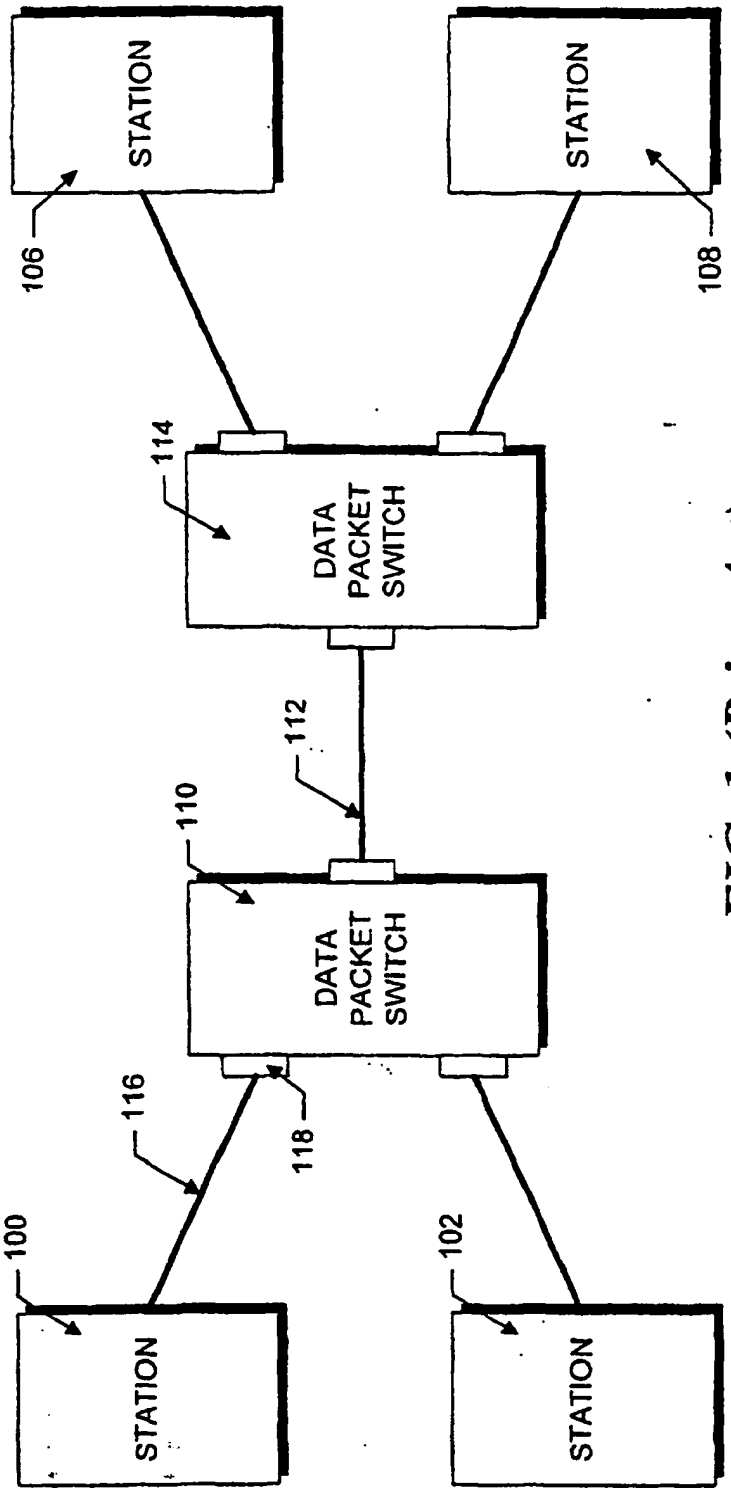

FIG. 1 (Prior Art)

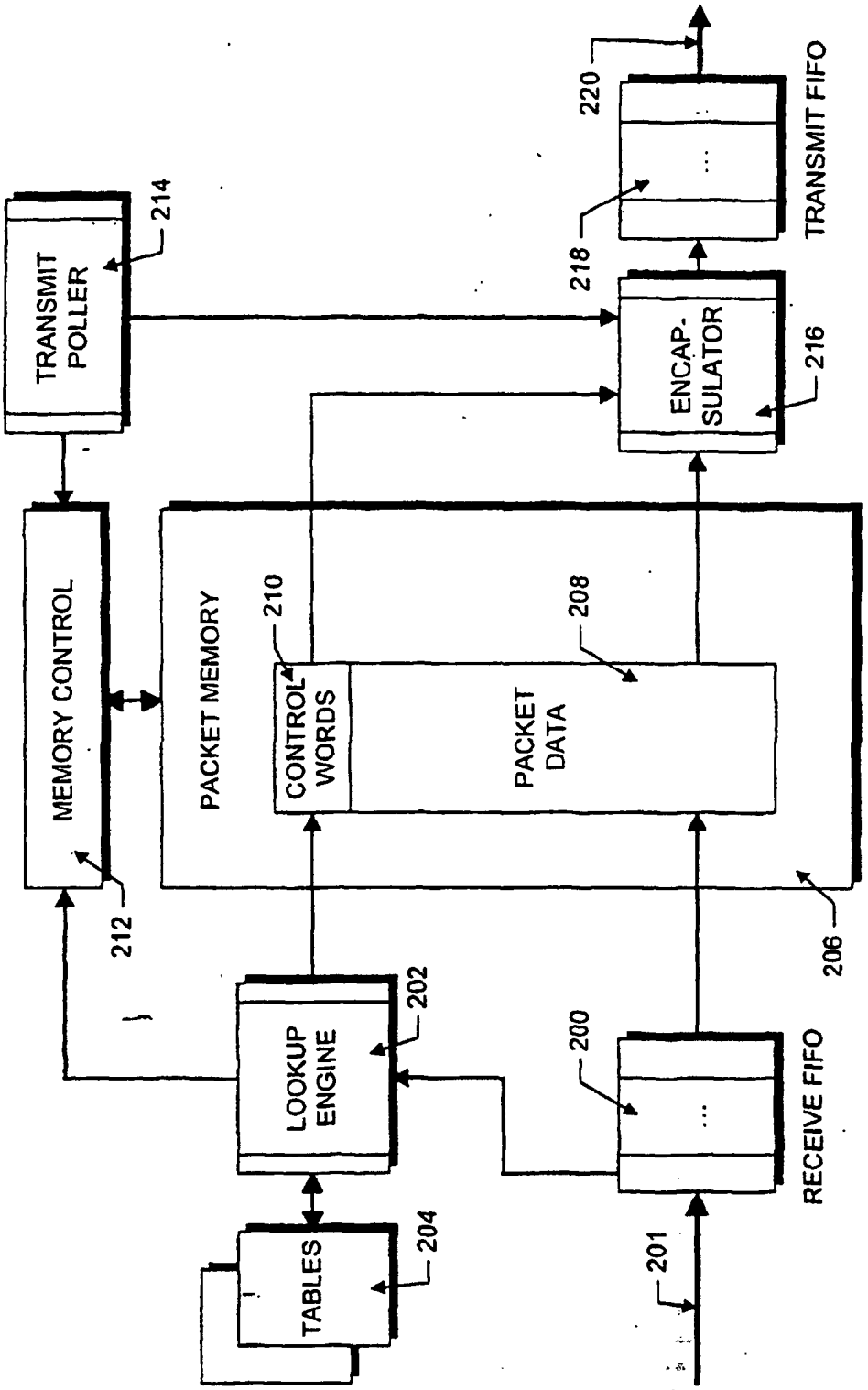
TRANSMIT POLLER
214
MEMORY CONTROL
212
PACKET MEMORY
CONTROL WORDS
210
PACKET DATA
208
ENCAP-SULATOR
216
218
220
TRANSMIT FIFO
LOOKUP ENGINE
202
TABLES
204
200
RECEIVE FIFO
201
206

FIG. 2

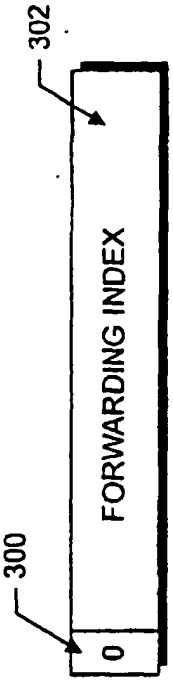
300
302
0
FORWARDING INDEX

FIG. 3A

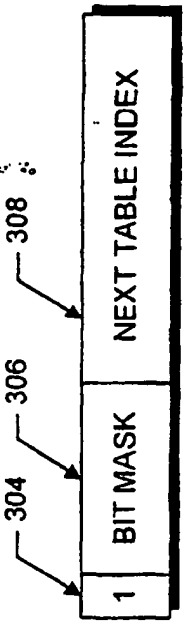
304
306
308
1
BIT MASK
NEXT TABLE INDEX

FIG. 3B

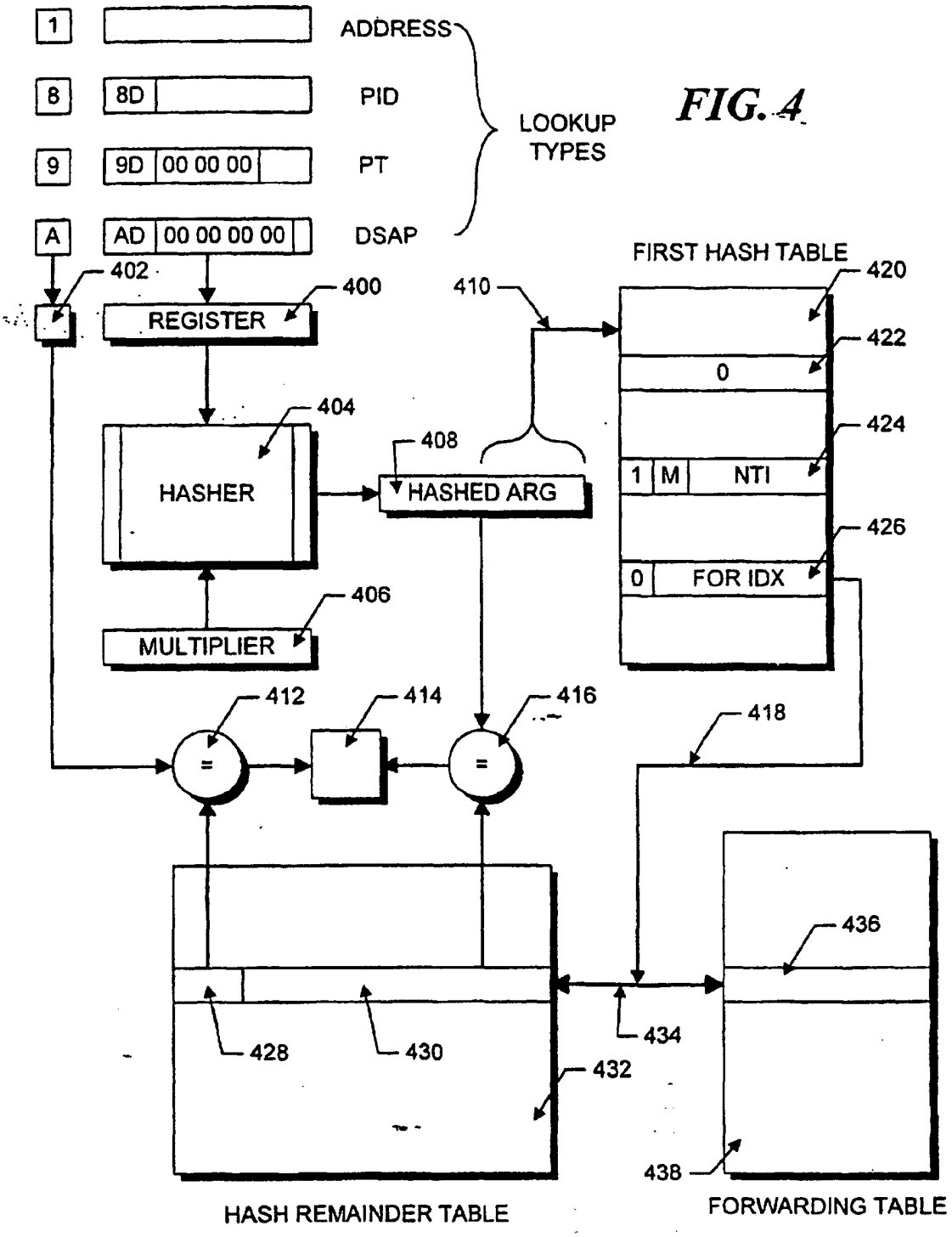
1
ADDRESS
8
8D
PID
9
9D
00 00 00
PT
A
AD
00 00 00 00
DSAP
LOOKUP TYPES
402
400
REGISTER
404
HASHER
406
MULTIPLIER
408
HASHED ARG
410
FIRST HASH TABLE
420
422
0
424
1
M
NTI
426
0
FOR IDX
412
414
416
418
=
=
428
430
432
434
436
438
HASH REMAINDER TABLE
FORWARDING TABLE

FIG. 4

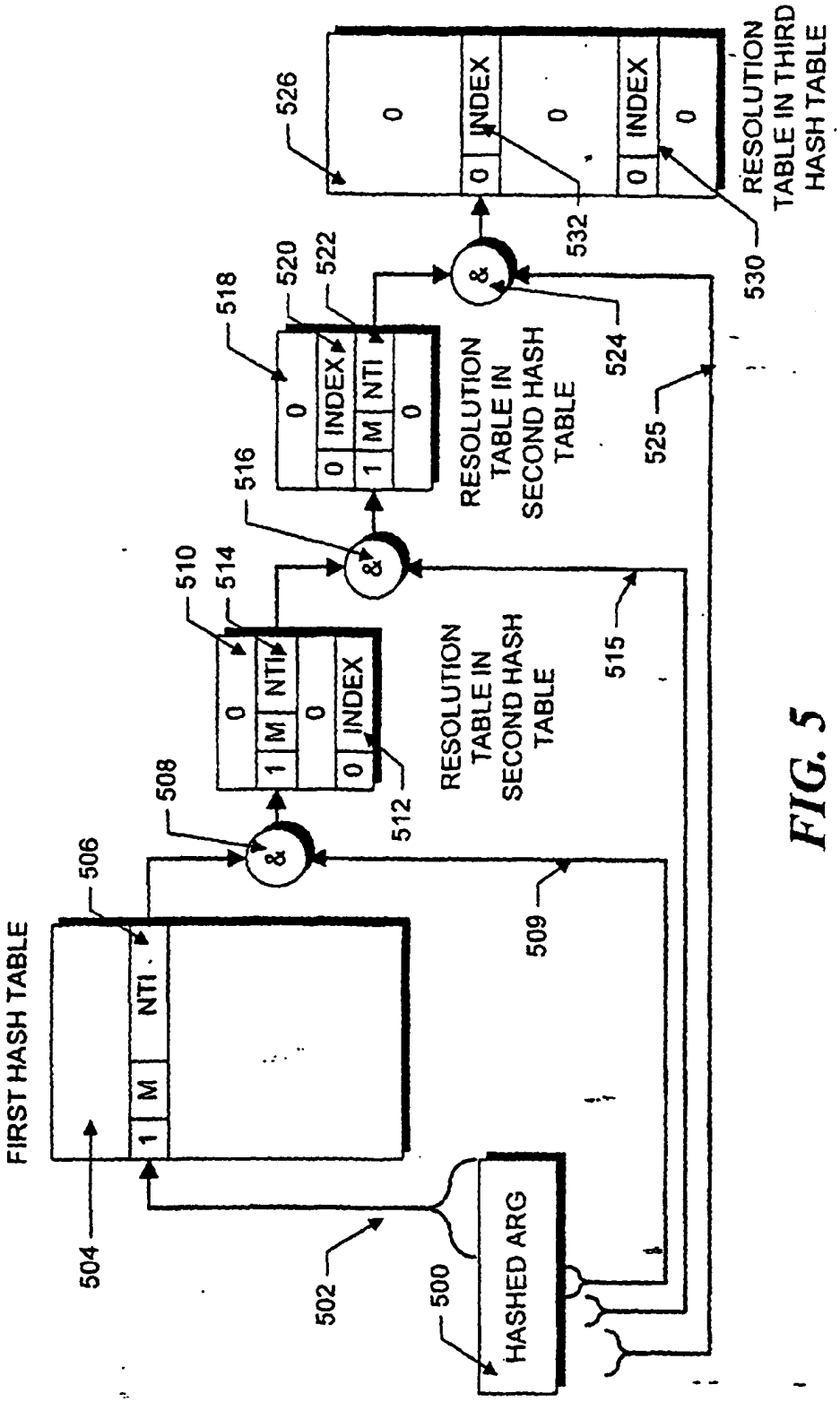
FIRST HASH TABLE
HASHED ARG
RESOLUTION TABLE IN SECOND HASH TABLE
RESOLUTION TABLE IN SECOND HASH TABLE
RESOLUTION TABLE IN THIRD HASH TABLE
INDEX
NTI
M
504
502
500
506
508
509
510
512
514
515
516
518
520
522
524
525
526
530
532

FIG. 5